The DFAS Test Professional's Companion

Introduction

EVERYTHING ABOUT TESTING IS ITERATIVE- This is an underlying testing principle. Appropriate testing accompanies each step of software and system assembly and is repeated with each change. Testing begins with the definition of requirements and continues throughout the life cycle as requirements are defined and refined; code is written, tested, and modified; components are assembled and tested together; end users and interfacing systems are added and their integration tested; the system is deployed; and modifications are proposed, made, and tested. This document is a supplemental discussion of testing issues, concerns and techniques. It provides lessons learned and guideposts, but is not a step by step cookbook.

A wealth of information to help with test planning and management is located on the Test Integrated Data Repository. Access is limited to those who support or review DFAS programs; if you qualify, you may obtain a user id and password by contacting Brian Hill at the Joint Interoperability Test Command (JITC), ***hillb@fhu.disa.mil***. The DFAS standard test methodologies and procedures for developing test plans and necessary test documentation are available on the Infoweb. Look under ***https://infoweb.dfas.mil/technology/pal/*** and check the PAL index to find what you need. Also on the web are the System Development Scenario (SDS) and System Modification Scenario (SMS) which integrate test activities with other necessary system development or modification steps. From the DFAS public home page, go to the Reference Library and look under "Program Manager Tools", ***http://www.dfas.mil/library/***. Finally, the web has the DFAS guidance on testing included in the DFAS 8000.1R document. Look on the Infoweb and navigate to policy and procedures, then to regulations, ***https://infoweb.dfas.mil/library/Regulations/***.

Please note, the Uniform Resource Locator (URL) addresses used throughout this document, reflect currently accurate addresses. These addresses may change or be updated. Also, some restrict access to specific users.

Why We Should Care- There are major benefits to a good testing program. Three very important ones are the reduced cost of development and implementation, improved customer satisfaction (public relations), and reduced cost of operation.

Development costs and schedule are often hostage to delays caused by the discovery of errors or major oversights late in the development cycle. Extra resources and overtime drive up costs in conjunction with the need to back up to the point of the omission, fix it, be sure it works, then try to catch up to original projections. A good testing program provides opportunity to discover and fix these problems when they occur. The program stays on track and the extra costs of extraordinary effort and schedule slips are avoided.

Often, the acceptance test is the customer's first opportunity to test-drive the new product. If careful testing and strict readiness reviews have not been enforced, the acceptance test will not lead the customer to accept the system. The acceptance test becomes the forum for discovering data initiation problems, data transfer and interface

problems, mismatches with specifications, computational error, problems in accessing system information and features, and so on. The experience will lead the customer to reject the system and make the customer suspicious of the system forevermore.

Perhaps worse is the consequence when insufficient testing leads to deployment of a flawed system. Common problems are abysmal response times, dysfunctional interfaces, missing or incorrect initial data values, essential reports that cannot be produced, essential processes that cannot be executed, and users left clueless by inadequate training and documentation. These problems lead to costly and time-consuming workarounds while problems are isolated and fixed, lost productivity, and risk that misstatement of information in critical, but incorrect, reports may lead to major organizational blunders. It goes without saying (because the customer is going to say it for us) that this is a most unpleasant experience for the customer. Good, balanced test discipline and a happy customer are a much better alternative.

When flaws are not as obvious, customer impact may not be so great. However, any post implementation flaws will lead to increased production costs. Statistics show it is many, many times cheaper to find and fix a problem in early testing, than it is to find and fix that same problem in production. This is one of those statements that makes so much inherent common sense (less time, fewer people), it is difficult to understand why testing frequently gets the proverbial "short end of the stick" in terms of enforced discipline, resources, and scheduling during the development cycle.

Goal- The primary goal of testing is to ***find those pesky errors and take the earliest opportunity to eradicate them.*** The importance of this cannot be over emphasized. A good test finds errors. Testers must be attuned to the concept that their job is to break the software. The conclusion that the software works can be drawn when our best efforts cannot prove that it doesn't. Developers and program managers will prefer tests that prove the software works, with ancillary discovery of errors leading to the unwelcome conclusion that it doesn't. This conflict of prime objectives is the reason many guidelines call for independent testing from the software integration test through operational test and evaluation. It takes a certain creativity to envision ways in which the software may fail. This creativity can be stifled in those who prefer to see everything work as planned.

Even tests with the objectives to verify requirements and validate that the software meets them are designed to find and eliminate flaws. Performance tests (when I run the software under expected conditions, will performance degrade unacceptably?), compliance tests (are there deficiencies that will keep the software from obtaining appropriate certifications?), and acceptance tests (are there customer requirements that have not been satisfied?) fall into this category.

Principles-How much is enough- Testing sufficient to guarantee that every possible error has been found and corrected would take infinite time and money. The software would never be fielded. Insufficient testing leads to enormous expenditures of time and money. The software may never be fielded. Obviously, somewhere in the middle is the happy medium—testing that is ***just right***. Helping us to find that happy

medium is the discipline of risk assessment. We determine which requirements are most important and determine the impact of potential failures. Knowing this, we figure in how much risk we are willing to accept. This tells us how much testing is enough.

Principles-Get the tester involved early- The program manager and customer must ultimately decide the proper priority of requirements and the impact of failures. They must also agree to what level of risk can be accepted. The tester cannot do it for them. The tester can help, ***but only if*** the tester understands the requirements themselves and how the associated priorities and risks are represented in the software so that testing can be appropriately concentrated. For the tester to attain full understanding, and provide maximum value, he/she must participate in the development from inception: requirements definition.

Principles-Agree on requirements and create a requirements traceability matrix- Involve testers in requirements definition, interpretation, and reviews. The test team will make sure requirements are stated in non-ambiguous, testable terms. These requirements become the core of the requirement traceabiblity matrix (RTM). The RTM, in turn, is the basis for tracing each requirement to an associated test and test result, and is thus the basis for demonstrating that requirements were satisfied. While there is no mandated form or format for the RTM, several DFAS systems have used them successfully. The program test directors for SABRS, DCAS, DIFMS, and DJAS can provide good examples of RTM that may be applied to other systems.

Principles-Prioritize requirements- Program managers and customers need to agree on the priority of requirements. While this requires serious effort, the agreement will be invaluable during development and testing. Time and dollar constraints may dictate that some requirements be dropped from the initial release. The priority of requirements makes such decisions easier. The priority plays an early role in test planning by providing the basis for the risk analysis that defines the scope of the test. The priority plays a role again when decisions must be made to curtail testing once it has begun. Below is a link to one discussion about prioritization techniques and issues. There are many other discussions on the web. Search and review several in light of the needs of your project. Just remember to establish a consistent method for your project and that if everything is "top" priority then nothing has been accomplished. In addition to a reasonable method of prioritizing the requirements there must be an equitable means of distributing them across the requirement categories.

http://www.processimpact.com/articles/prioritizing.

Principles-Plan for success- Testing must be defined and planned up front. Program managers must identify who will provide test support and agree early with developers, users, and testers on test roles, schedules, and resource requirements. The Program Test Director is the primary focal point for this early planning and coordination. Planned test strategies, methodologies, participants, resources, and schedules are the basis

for the Test and Evaluation Master Plan (TEMP) and the more detailed supporting plans for each test event. The TEMP provides the testing overview for the program and also serves as a contract among the major test stakeholders. The more detailed supporting plans lay out tests to be conducted at specified points of the software life cycle and are a critical component for success. "Seat of the pants" or ad-hoc testing has been proven over and over to be a poor practice. Software is too complicated and too critical for the testing process not to be well thought out ahead of time. Proper test plans ensure that everyone knows what they are trying to accomplish, what must be done to accomplish it, and how they will know when they have completed the tasks.

And like the test itself, plans are evolutionary and iterative. All test plans are subject to several updates before test execution. An important entrance criterion for each test is to be sure that the appropriate plan is in its final form and approved before the test begins.

Principles-Documentation- In addition to the planning documents that must be prepared, the tests themselves must be documented. Documentation provides essential test roadmaps and audit trails. Planning documentation describes what must be done and what results are expected. Test execution documentation provides evidence that the test was conducted, evidence of what the software did, evidence of the result, evidence of discrepancies and errors. It provides the basis for analyzing results to determine whether software performed as expected and whether compliance mandates were met. It provides the basis for successfully repeating the test once changes have been made so that we will know: (1) whether the problem was truly fixed or (2) whether a fix or change caused a problem somewhere else.

Principles-Regression testing- Since our primary premise is that testing is an iterative process, it follows that we must be able to repeat tests. Tests must be repeatable because they must be used over and over as software is developed, fixed and modified. Within each cycle or release, regression testing will ensure that all the software that previously executed correctly still executes correctly. It is fairly common for changes to software to have unanticipated effects on seemingly unrelated code. Testing literature relates numerous accounts linking the omission of regression testing to software disasters and consequential schedule and cost disruptions. Better we learn from another's mistakes than our own and plan for the necessary regression testing. As with initial testing, the level and completeness of regression testing must be based on a risk assessment. The complexity of the code, the difficulty of the intended change and the impact of any problems must be evaluated. Also consider the maturity level and track record of the development organization in such risk assessments.

Major DOD Test Categories- DoD categorizes testing as either developmental or operational. This handbook is largely limited to a discussion of developmental testing.

In developmental testing, software is tested in the development hardware environment as it is gradually integrated through the building and testing process until the whole application/AIS/system is tested as an integrated whole. Developmental testing validates

the complete system in the development environment, including the use of all expected external interfaces. Developmental tests should evaluate everything that might impact successful operation once the software is released, including training, operations manuals, performance, and restart/recovery. As the software is developed and tested, it gains more and more of the attributes it will have in its final form. Accordingly, tests are conducted on platforms and using data that is increasing similar to the anticipated production environment. How much like the planned production environment the last phase of developmental testing, the software acceptance test, should be is best determined through risk assessment. The more complex the system, the greater the risk that an environment-related problem may go unanticipated; acceptance testing for such systems should mimic the production environment as closely as possible.

The results of operational test and evaluation (OT&E) provide the basis for the final deployment decision. Ideally, the software is deployed at a limited number of sites where the operational test organization can evaluate all aspects of its effectiveness and suitability to satisfy the originally defined mission need. When it is not practical to release the system at a prototype site, a parallel production site, or a limited number of initial sites; the OT&E may be performed using a simulated production environment with people, training, and interoperability issues mimicking the real thing as closely as possible.

Levels of Developmental Testing

Unit testing is applied to an executable chunk of code. This chunk can be defined differently in repositories or the object oriented environment. In simpler times, the unit was defined as a program.

Unit testing is based primarily on white/glass/structural viewpoint planned against the program specifications. Programmers execute unit tests using the equipment they use for development. While unit testing does not require extensive documentation, the test cases that were executed should be recorded, as specified in DFAS standards, so that these can be repeated as necessary for regression testing. Unit test data does not require functional relevance, merely that the data mimic valid (or invalid) transactions. For example, unit tests may use appropriation numbers that do not exist, as long as no format rules are violated. Programmers may also substitute names such as "Wiley Coyote" or "Mickey Mouse" in lieu of those more likely to represent a living person.

System level testing happens in many phases, each with its own name. System level tests typically include tests with such names as the Software Integration Test, the Software Qualification Test, the System Integration Test, and the System Qualification Test. At DFAS, we are developing the DFAS Corporate Information Infrastructure (DCII), in which several upgrades and releases of corporate applications are incorporated in phased DCII releases. To handle this extra level of integration, we have replaced the traditional SQT and SAT with the Functional Validation Test, the Integrated Functional Validation Test, the Enterprise Integration Test, and the Enterprise Acceptance test. There is also an Enterprise Performance Test. In all cases, the objective is to test the whole system as one

entity, validating all the requirements that can be validated in development environment. It is important to remember that things like data load processes, data conversion processes, operational processing requirements, manuals, all technical requirements (memory usage, restart recovery etc.) should be tested at the system level. It is insufficient to limit the test to application software, fudging how the data gets in the system and ignoring external components. These external components will invariably foul up in the long run if potential problems are not anticipated by testing. Emulate the production environment as closely as possible. If possible, use the same versions of software and hardware. With the exception of performance testing, which will simulate production levels, data used for system level testing will not equal the amount that will be processed once the system is in production. However, the test must encompass the end-to-end processes that are anticipated once the system is in production, including data load, data conversion, and the initial population of data tables. The system level tests give the opportunity to identify environmental or externally driven problems not directly related to the new software being produced.

Develop and update test plans in conjunction with the increasing knowledge gained from the various progress reviews (e.g. requirements reviews) that occur during development. Plan to use the test plan to guide test execution and document it accordingly. The necessary test processing flow and execution sequence becomes clear as testers gain functional understanding of the system. The flow or sequence used may represent only one of many valid ways to execute the process, but it is generally better for development testing to be single-threaded. Processes may be tested concurrently, but only where one sequence does not impact another. This control pinpoints processes that are the root of problems by removing any doubt that an intertwined process may be the culprit. In addition, the discipline will facilitate regression testing-- the same thing executed in the same order should produce the same results. When you have completed formal, structured testing, it is generally beneficial to give the users an opportunity to develop their own "ad-hoc" tests and run them. The focus shifts to the user's ability to use the system and the system's ability to satisfy the mission. At this point testing should no longer be single-threaded.

The first of the system level tests (SIT) should validate documented requirements, both functional and technical. The test should evaluate system compliance with various mandated standards (overarching requirements) and as many quality assurance (QA) attributes (maintainability, portability, usability) as practicable. Refer to the program Operational Requirements Document (ORD) for any of these not explicitly identified in the requirement baseline. For DFAS DCII systems, this level of testing occurs during an event called the Enterprise Integration Test. The earlier SIT test conducted for each component will not have interacted sufficiently with other system components to comprehensively address overarching requirements and QA attributes. Comprehensive or not, however, these should still be evaluated and not left to later tests. Once again, the earlier a problem is identified, the easier and cheaper it is to fix.

Tests such as Software Qualification Test (SQT), Functional Validation Test (FVT), Software Acceptance Test (SAT) and Enterprise Acceptance Test (EAT) validate that the

system suits the user and satisfies the mission. These tests may resemble those run during SIT and may even use the same data; however, they are designed and evaluated from the user's perspective. The goal is to ensure that the system fulfills user business process needs. While conscientious effort may have been made at program initiation to accurately represent these needs, we may not have succeeded in capturing all of them in our requirement baseline. These user-oriented tests provide the user a first opportunity to test drive the system and this is where such discrepancies will be revealed. Involve the end-users as much as possible. They are the people who understand their day-to-day business process and can evaluate the system software against those needs.

On the other hand, users may identify as discrepancies things that were not originally envisioned as part of the system solution. These may range from conveniences to things that will yield great productivity savings. Whether these should be categorized as things that must be fixed or added before the system can be accepted or things that can be incorporated in a later version of the system must be negotiated. A review team including testers, program management representatives, customer and/or user representatives, and developer representatives needs to review and agree on the priority and category of each identified problem. Without such a disciplined process to review problems and document disposition agreements, the user may never be willing to formally accept the system, and acceptance testing will never end

Test Techniques-

White box, glass box, and structural tests require access to source code, are based on code, and probably will require stubs and drivers to emulate the external environment. There are many references describing various standard and accepted approaches to unit tests. These standards need to be translated to whatever development environment is being utilized. They include guidelines such as approaches that can be taken to build test cases. The references provide benchmarks for statement coverage, decision coverage, branch coverage, path testing, cause and effect graphing, and boundary value analysis. Several references are listed at the end of this document.

Black box and functional tests provide visibility of what's happening in the application by examining the data going in and results coming out of the tested component, and determining whether the results are as expected. The tests do not examine the path the data takes through the code; in fact, the tester has no visibility of the code. Again, there are many references providing guidelines and approaches that must be interpreted in light of the development environment. Client-server environments provide added complexity since all the interactions between various pieces of software residing on different components must be considered. Several references are listed at the end of this document

Test Environments-

Programmer initiated tests (Unit) are done in the development environment, ideally one area for each programmer.

System level tests are also done in the development environment, matching as closely as possible all operating and management software versions, and the hardware and software configuration items that will be used in production. The closer the development environment can be made to match the production environment, the greater the chance errors will be identified and the fewer the unanticipated problems to be found in production. As software is developed and integrated with the total set of components to be packaged in the release, the test environments should become more and more like the anticipated production environment.

For software to be deployed in a client server environment, the ability to mimic the production environment is particularly important, and particularly hard to do. The combinations of local area networks, wide area networks, communications paths between components, firewalls, operating systems on clients and on servers, web servers and browsers, are impossible to anticipate. Still, it is important to emulate as much as possible the targeted environment.

Organizing and Planning the Test-

There are three major considerations in organizing and planning for the test: execution steps that emulate those anticipated in production, test data that represents anticipated variations and problems, and an adequately configured test environment.

Know what is most important to test. Use the priorities defined by the users in early requirement definition. If the priorities have not been defined, develop them now. Without defined priorities, testers will prioritize based on their best knowledge and understanding of the system. This is risky and a poor substitute for the more accurate priority subject matter experts, customers and users can provide. Once the priority is established, categorize the requirements into business processes (scenarios) to be tested. These functional scenarios will then be amplified to include assessment of the systems ability to meet technical requirements for SIT. When planning the execution steps and sequence of events for system level tests consider:

(1) What system processing cycles need to be represented in the test- daily, weekly, yearly, etc? How does on-line processing fit in? Is there batch processing to be represented? ***REMEMBER***, you want the test to be repeatable. Some processing is not dependent on other events and may be executed in any order or simultaneously. However, in order for the test to be the same each time, a specific processing sequence must be defined and adhered to.

(2) What is the proper ordering of business processes and program execution within these processing cycles?

When determining what data to use in the test consider:

(1) What is the source of data should be used in this test? "Real" production data may be composed of thousands of transactions that may or may not represent all the

conditions to be tested. If production data is to be the basis for test data, it may need to be culled to reduce volume and supplemented to represent all possible conditions.

(2) If there is no source of production data from which to generate test data, the initial test data will have to be built. This task is generally difficult and time-consuming. Testers may have to rely on subject matter experts to create and provide the test data, which may have unanticipated schedule impacts. There are special tools that will facilitate the test data creation task.

(3) What is the proper size of the test data file? When determining the size of the test data file, consider the time it will take to validate transactions. It is important to evaluate the loading or rejection of data, but it is also important to determine: whether transactions were loaded without rejection, whether transactions were transformed into invalid or valid but different transactions by the load, and whether all transactions not specifically rejected were loaded. A strategy for evaluating the latter circumstances must be developed and the size of the data file directly impacts how comprehensively this analysis can be done.

While testers may not exercise full control over the test environment, the following is important for documenting the test plan:

(1) Document all pieces of the environment, what they are, what software and version is to be utilized.

(2) For tests other than performance, sizing of items such as the processor, communication links, and data storage may not be critical; the test data will not require capacity equal to that required for true production. However, the extent to which such sizing will impact testing should be carefully evaluated by database staff, technical staff, and testers to be sure the test environment is adequately sized. Capacity should not impact or confuse test results. For example, be sure there are sufficient rollback segments even in the test environment.

A Word on Scenarios and Scripts

Scripts are the instructions/procedures for executing a specific test condition and evaluating the results. They must include the set-up, execution and validation procedures.

Scenarios are collections of scripts that test a business process. They must be created and then mapped to the execution sequence. They should be predicated on the priority and criticality of the processes

Test Cases are a statement of what is to be tested. At the system level they essentially state the requirement according to how it will be implemented and measured.

Event Log- Event logs are important records of the test. Refer to the test standards and procedures referenced above for formats and guidance on establishing and keeping event logs.

Test Report- Test reports may be written on several levels. Reports may be created for organization executives, the Program Manager, the Program Test Director, the development organization, or the testers themselves. The reports will differ according to the required level focus and specificity. For example, management reports will contain summary level information while the development organizations will need information to allow focus on the sources of problems and cures for them. Use referenced standards and procedures for mandatory formats. Some reporting requirements may be new or unique, so be sure to define types of reports to be created and formats before testing begins.

Resourcing the Test Team- Determine the number of dedicated testers according to schedule and simultaneous test-related events. A good rule of thumb is to assign one tester, full time, for every four developers. They need to be assigned early and should participate in early requirements definition and review sessions. Test planning starts with these requirements defining sessions at the initial stages of the project. The test team should be highly skilled with a mix highly proficient in both the technical aspects of testing and the functional aspects of the business requirement to be addressed. Well-rounded professionals who exhibit initiative and creativity are a must. Some testers may be taken from the development staff to provide a realistic perspective on the difficulty of testing and a better understanding of what happens when all the code comes together.

Boris Beizer in his book Software System Testing and Quality Assurance suggests these nine qualities for the testing professional:

1. Experience as a programmer- for technical members
2. A thick skin and a good sense of humor- absolutely essential or they won't survive
3. Tolerance for chaos- testing surely is that.
4. Tenacity- they must not give in when they see a problem
5. Skeptical- questioning everything for potential problems
6. Cunning- that is wily and skillful.
7. Bold- not easily intimidated.
8. Self sufficient- love doesn't come from the job, need good self-esteem.
9. Honest- want the best possible product and don't like compromise.

Add to this mix the quality of a strong backbone and extraordinary skill in organizing work and you have the makings of a world-class testing team

Basic skills are not unlike those required of most professionals in today's automated world: good PC office software skills that include word processing, spreadsheets, project management, and single use databases. In addition, the test team should already know how or be trained to use the test tools and software with which they will manage and execute tests. In DFAS this includes the Mercury products Test

Director and WinRunner. Those who participate in performance testing will also need to be able to expertly use LoadRunner. For DFAS, experience with Cognos and Impromptu along with a basic understanding and ability to work with SQL are essential. Bonus points are awarded for the ability to use the evaluation software TOAD. Those who posses a general understanding of whatever development tool are being used receive extra bonus points..

Training for the Test Team- Members of the test team must be trained in basic test principles as well as in the specific tools and techniques to be used. There are several basic courses available, either classroom or over the Internet, from the Software Testing Center. Tuition is less for training taken over the net, and individuals can schedule such training so that it will not intrude with work schedules. The classroom training is also reasonable and, if several testers need the training, may be an excellent opportunity to begin the team building so important for success. The Software Testing Center URL is http://www.testingcenter.com. As mentioned earlier, client-server testing includes techniques not necessarily encountered during traditional mainframe--based testing. The Client Server course offered by the Software Testing Center is a good introduction to the different approach needed.

For the future, DFAS is developing a test class focused on applications developed using Oracle Designer/Developer and based on functionality from the DCII. This will be particularly valuable for DCII testers. It is planned to be available in spring, 2001.

In addition, there are numerous conferences, seminars and other courses available. A web search will reveal just how many of these are available and will help locate the right one.

In addition, there are several professional testing certifications. Perhaps the best known the Certified Software Test Engineer conferred by the Quality Assurance Institute of Orlando, Fl. Also available is the Certified Software Test Professional designation offered by the International Institute for Software Testing of Inver Grove Heights, MN. It is highly recommended that Program Test Directors hold or enroll in a course of study to obtain such certifications.

Tools- Standard software, software tools, and software approved for execution of the DFAS ELAN are enumerated on the Infoweb. They can be found at these URLs:

https://infoweb.dfas.mil/technology/infrastructure/standards/sw/corplice.htm

https://infoweb.dfas.mil/library/regulations/hq/80001r/bc2ap4.pdf

https://infoweb.dfas.mil/technology/infrastructure/standards/sw/appsftw.pdf

While many of these are not test specific, they can facilitate the work of the tester, for example, drawing tools can be sued to diagram system flow processes and test environment configurations.

Standard test tools include the Mercury Test Suite and the Configuration management Information System (CMIS) for recording and tracking Test discrepancy Reports.

There are also data generators, file compare tools, tools to measure software performance. These tools can help manage and improve testing. Some of the metrics tools can provide especially useful statistics to measure software performance and errors detected and thus demonstrate how effectively tests identified and eliminated errors..

General Resources and References- Every test location should have a library of test references. Supervisors and testers should work together to develop development programs that encourage the testers to expand their field of knowledge. Information about software testing continues to grow and new techniques are developed and discussed daily. Testers should consider it a part of their job to remain current in their discipline and to independently pursue requisite knowledge.

Seminars- Testers should plan to attend one testing conference a year to share their experiences and to learn what other organizations are doing. These seminars also introduce the latest testing technology and disciplines. The annual conference of the Quality Assurance Institute held each fall in Orlando FL, and is a rich source of good information, especially for beginning and intermediate testers. Other worthwhile seminars include the annual STAR East and West conferences. Software Quality Engineering, out of Jacksonville FL, offers many especially challenging seminars. Those interested should search the Web every month or so to discover the most recent offerings.

Books offer readily available, comprehensive references. Each test site should acquire and keep current a basic library of testing books. The following is a suggested starting bibliography:

Myers, Glenford	The Art of Software Testing- classic in the field written for the beginner. Very useful
Beizer, Boris	Software Testing Techniques- these are written with humor and guaranteed to keep the reader awake and thinking Software System Testing and QA Black Box Testing
Caner, Kem	Testing Computer Software- excellent text with list of 400 common bugs in IBM software. This author is also an attorney and very active with Internet security concerns..
Siegel, Shel	Object Oriented Software Testing- written specifically for object-oriented applications.

Magazines- Software Testing and Quality Engineering and Crosstalk are good testing reference periodicals, which should be included in the periodical section of the reference library. Magazines devoted to software development will also contain useful articles on test and evaluation.

Web Sites- Some specific sites for testing-related information are listed below; however, the information available on the web changes rapidly. A web search (the search engines "Google" and "Alltheweb" are particularly recommended; "Dogpile" and "Northern Light" are also good) for the words "software testing" will return a wealth or informative sites.

http://www.evolutif.co.uk/ This is a European link well worth exploring. Particularly look for their white papers. Very good discussions of lots of issues.

http://www.ondaweb.com/sti/ - This link is to the Software Testing Institute located in Dallas. It was founded by the woman who first started Autotester, an automated tool.

Some search sites, such as http://www.karnak.com , offer subscriptions to search the web periodically for specific items of interest and provide periodic updates.